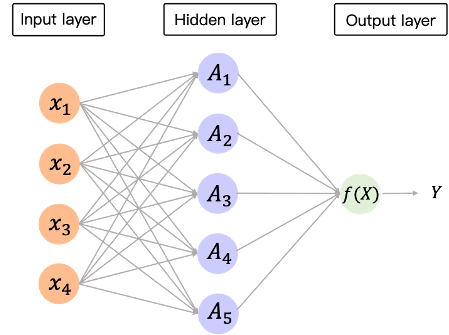
Neural Network （NN；神經網路）

神經網路即模仿人類腦部神經元發出訊號、運作、做出反應的模式來建立模型，並且可以處理迴歸與分類等問題，利用解釋變數或特徵當作輸入層（input layer），中間經過非線性方程式構成的隱藏層（hidden layer）轉換後，預測反應變數當作輸出層（output layer）（圖一）。其中，輸入與輸出層僅有一層，「層」是由多個節點組成，節點之間的連結與權重、門檻值有關，若節點高於門檻值則可以傳送至下一層（隱藏層），至於隱藏層的數量則無限制，每一個隱藏層都會分析處理前一層的資料，再傳遞下去。基於模型會自主學習，最終輸出的結果就是已訓練完成的網路，中間的過程則像黑盒子一樣，層與層之間的關係無法利用方程式描述；此外，參數的設定則影響輸入層與輸出層之間的擬合，最佳的結果是不斷重複試驗參數獲得之。



圖一、神經網路示意圖（https://ithelp.ithome.com.tw/articles/10305927）。

神經網路類別有DNN、CNN、RNN、LSTM：

1. DNN（深度神經網路）：即是NN模型的複雜版，隱藏層不只一層，當層數越高，所處理的問題複雜程度也越高。
2. CNN（卷積神經網路）：專門解決圖像問題（空間），可以當作獲得照片的局部訊息，多個局部訊息拼起來就是整張圖的特徵。
3. RNN（循環神經網路）：專門建立語言的相關模型（時間），因為必須考慮前後文的意思以避免斷章取義。以下面兩個例子為例：
   * 我不吃辣，所以，我點「蝦仁豆腐」
   * 我吃辣，所以，我點「麻婆豆腐」

若僅考慮「我點」這個單詞時，蝦仁與麻婆豆腐的機率是相等的；反之，若考慮「我吃辣」，選麻婆豆腐的機率會變高。

1. LSTM（長短期記憶模型）：改善RNN在長期記憶的不足（權重消失、梯度爆炸），並且會選擇性忘記—忘記不重要的，記住重要的，長期訊息可以幾乎不受影響的傳遞下去。

神經網路的基本訓練流程：

1. 定義神經網路，設置學習的參數或權重

即 初始參數設定（hyper-parameters）、nn.Sequential()的建立

1. Training data的設定與迭代

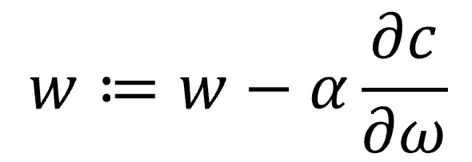
e.g. 資料切分成training/test data，以training data訓練模型

1. 神經網路開始處理input layer
2. 計算Loss function，也就是預測結果與正確答案的差距

\*loss function設定詳細見：損失函數––函數

1. 藉由Loss function得到新權重

新權重=原權重-learning rate\*梯度



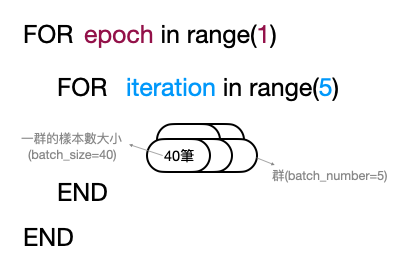
1. 更新神經網路權重

利用優化器（ Optimizer ）去更新

1. 重複步驟3-6，直到找到一組權重使loss function最小。

**Torch.nn 參數設定**

1. **初始參數設定**
2. Epoch: 訓練週期
3. Batch\_size: 一群樣本數大小
4. Batch\_number: 共有幾群樣本數

e.g. 200筆資料，每一群的大小（batch size）有40筆資料，共有5群（batch number），每一輪要學5群資料，也就是5次迭代（iteration），學完5個迭代後就是一輪epoch訓練的結束。

1. Learning\_rate: 學習速率，控制梯度下降的速度，太大會造成模型不穩定，太小會學習速度太慢，其值通常小於1。
2. **nn.Sequential() 的建立**：快速搭建「一層連接一層」的模型結構，並一次執行forward propagation（即輸入模型輸出的過程）。模型結構通常包含：學習函數、激活函數、池化層。

a. 學習函數：根據資料來選擇之。

e.g. (1) nn.Linear: 線性變換層

(2) nn.Conv1d: 主要用於文本資料、時序資料，只對寬度做卷積。

(3) nn.Conv2d: 主要用於圖像資料，對寬度與高度做卷積。

[Convolution in 1-dimension]

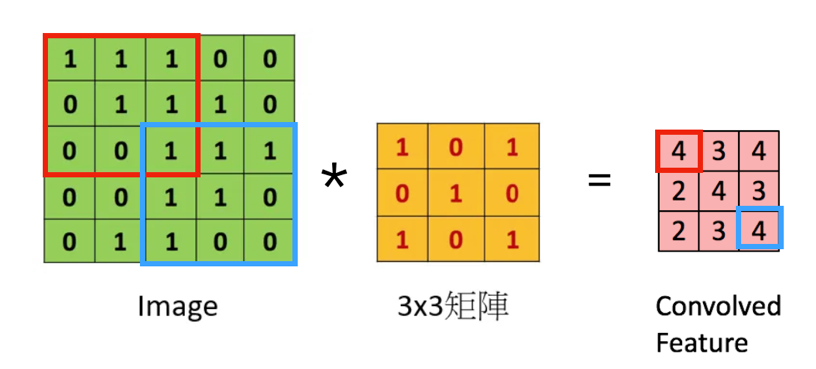
一個單一震源(source pulse)經過地球內部，所看到的訊號(signal)即是震源與地球間的卷積結果。

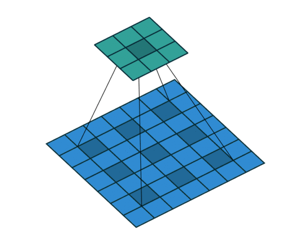


\*參數設定與Conv2d雷同。

[Convolution in 2-dimension]

3x3矩陣(kernel == filter)在image移動(stride)，計算兩個矩陣之間的乘積，得到convolved feature。



* Kernel\_size: 也就是上方橘色3\*3的矩陣，input經過這個filter即會得到3\*3的output。
* Stride: 移動步伐，kernel每次移動的距離。
* Padding: 在圖片外圍補0，避免feature減小。
* Dilation: 在兩兩卷積點中插入一個空白，3\*3變成5\*5（如右圖，dilation=2），保留更多細節但沒有增加計算量。
* Group: 將輸入分組，再對每組進行卷積。計算數量減少到原本的1/g。
* Bias: 可以想像成「截距」，偏移的部分可以更擬合資料。

1. 激活函數：為了讓網路學習更複雜的內容。假設沒有激活函數，output的內只是input內容的線性組合，因此，激活函數通常是非線性的。

e.g. nn.ReLU()、nn.tanh()、nn.sigmoid()

Comparison:

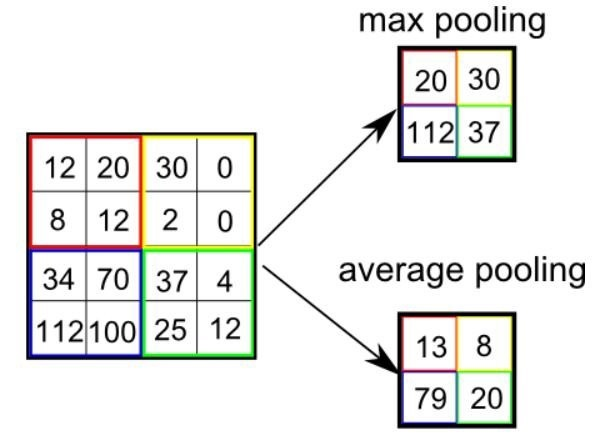
|  |  |  |  |
| --- | --- | --- | --- |
| 模型  名稱 | 函數 | 優點 | 缺點 |
| sigmoid | 輸出範圍：0~1 | a. 梯度平滑  b. 明確的預測，值，接近0/1  c. 常用於分類為題的輸出層 | a. 趨近0或1時梯度幾乎為0（梯度消失）  b. 輸出恆大於0  c. 運算慢 |
| tanh | 輸出範圍: -1~1 | a. 輸出以0為中心  b. 收斂比sigmoid快 | a. 梯度消失  b. 不利權重更新 |
| ReLU  [最常  使用] | 輸出範圍: 0~x | a.計算速度快  b. 改善梯度消失問題  c. 計算效率高 | a.輸入為負時，ReLU完全失效（Dead ReLU）  b. 不以0為中心 |
| Leaky  ReLU | 輸出範圍:0.1x~x | a. 改善Dead ReLU狀況  b. 具有ReLU的所有優點 |  |
| sofemax | 輸出總和為1 | a. 總和為1，較小的值具有較小的機率不會直接丟棄  b. 模型為互相獨立的類別，且只能選擇一個類別，採用這個函數較佳。 | a. 在零不可微 |

＊函數使用時機：

|  |  |  |
| --- | --- | --- |
| 問題種類 | 隱藏層 | 輸出層 |
| 迴歸 | ReLU | Linear |
| 二元分類 | Sigmoid/Tanh |
| 多元分類 | Softmax |

1. 池化層：也就是down-sampling，降低多餘的訊息、減少計算成本。像convolution一樣滑動，但每次的滑動不互相覆蓋，並以kernel涵蓋的最大值或平均值來篩選（如圖），而目前以Maxpool的效果最佳。

e.g. nn.MaxPool1d()、nn.MaxPool2d()、nn.AvgPool1d()、nn.AvgPool2d()



1. **損失函數（Loss Function）：**評估模型好壞的指標

a. 函數種類

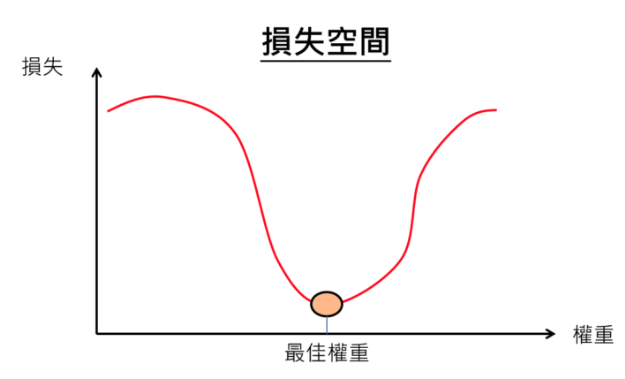
(1) CrossEntropyLoss（交叉熵損失函數）：分類問題，計算的是機率。每個類別的entropy越小，資料同質性越高，所有的entropy加總就是cross entropy，因此希望cross entropy越小越好。

(2) MSELoss（均方誤差）：迴歸問題，計算的是距離。

(3) MAELoss（平均絕對值誤差）：迴歸問題，計算的是距離。

b. backward() 反向傳播

反向傳播就是要以最快的方法走到損失最小的最佳權重。對應「輸出模型輸入的過程」，根據loss function的大小來判斷權重好壞。權重改變所對應的損失，即為損失空間（如圖），找到最佳權重使模型越來越精確。



1. **優化器（Optimizer）：**根據損失函數數值去更新神經網路權重

|  |  |  |
| --- | --- | --- |
| Optimizer | 解釋 | 特點 |
| SGD | 找出參數的梯度，往梯度的方向去更新權重（weight） | 需自行設定learning rate  長時間才能收斂至最小值  Loss function可能有嚴重震盪 |
| Momentum | 利用動量的概念，在同方向的維度上學習速度增加；反之亦然。 | 需自行設定learning rate |
| AdaGrad | 根據梯度調整learning rate | 適合處理稀疏資料集 |
| Adam  [最常使用] | Momentum與AdaGrad的結合 | 適用於大數據集與高維空間  目前最常使用的 |

通常Optimizer會與以下指令使用：

**output = cnn(bx)**

#資料放入模型，得到預測結果

**loss = loss\_func(output, by)**

#計算loss function：預測結果與真實結果的差

**optimization.zero\_grad()**

#梯度初始化為零：每個參數的梯度值皆清零；若不清零，Pytorch會將上次的計算的梯度與本次梯度累加。

**loss.backward()**

#反向傳播求梯度，將loss反向傳遞，計算當前梯度，找到最佳權重。

**optimization.step()**

#更新所有參數

**參考網站**

1. https://ah.nccu.edu.tw/bitstream/140.119/35873/6/25700606.pdf
2. https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db
3. https://finance.sina.com.cn/tech/2021-02-24/doc-ikftssap8455930.shtml
4. https://ithelp.ithome.com.tw/articles/10276865
5. https://zhuanlan.zhihu.com/p/463234293
6. https://hackmd.io/@allen108108/rkn-oVGA4
7. https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-ml-note-convolution-neural-network-%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-bfa8566744e9
8. <https://www.cnblogs.com/wanghui-garcia/p/10775367.html>
9. https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db